Université du Québec
École de technologie supérieure
Département de génie de la production automatisée

GPA789 Analyse et conception orientées objet

GPA789 Analyse et conception orientées objet

Professeur: Tony Wong, Ph.D., ing.

Chapitre 7

Notation UML

1

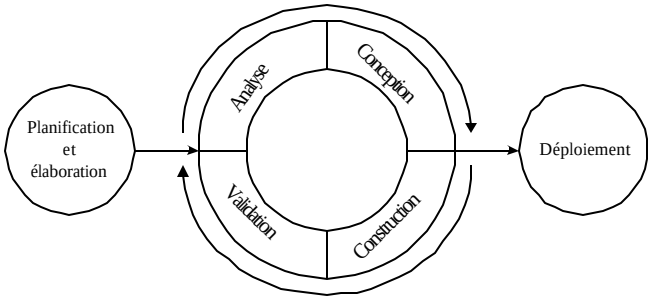
GPA789 Analyse et conception orientées objet

Introduction (1)

- Ce chapitre présente une démarche méthodique servant au développement orienté objet des logiciels.
- Le développement de logiciel comprennent nécessairement les phases suivantes:
  - La planification et élaboration
    - Il s'agit de rassembler les ressources humaines et matérielles, définir les exigences, construire les prototypes, etc. (Analyse et conception orientées objet)
  - La construction
    - Il s'agit de la réalisation du système logiciel envisagé.

Introduction (2)

- Le déploiement
  - L'application et l'implantation du système logiciel dans son environnement d'utilisation.
- Le diagramme montrant les phases d'un développement de logiciel



3

Planification et élaboration (1)

- Les tâches de cette phase d'activités:
  - Création d'un plan de travail
    - Le plan de travail doit expliquer le but du projet, les ressources nécessaires (humaines et matérielles), l'horaire des travaux (diagrammes de GANTT), etc.
  - Étude préliminaire
    - Il est nécessaire de prendre connaissance de la problématique. Il faut obtenir une bonne compréhension de l'utilité du système logiciel et de son utilisation. Dans la plupart des cas cela implique des rencontres avec les experts et les utilisateurs du domaine d'application.

4

Planification et élaboration (2)

- Description des exigences
  - Décrire en langage naturel ce que doit faire le logiciel. Cette description ne propose pas de solutions. Elle a pour but de présenter, d'une manière informelle, les tâches à accomplir par le système logiciel. (CAS D'UTILISATION)
- Dictionnaire des termes
  - Le dictionnaire est réalisé sous forme d'un glossaire expliquant le sens des concepts et les mots techniques utilisés dans la documentation de cette étape du processus de développement. Le dictionnaire sert à éliminer, dans la mesure du possible, l'ambiguïté du langage naturel.

5

Planification et élaboration (3)

- Le prototypage
  - Dans certaines situations complexes, il est nécessaire de créer des prototypes afin de saisir les nuances et subtilités inhérentes à la problématique. Ces prototypes peuvent être représentés par des maquettes sur papier ou encore par des programmes créés à l'aide de générateurs d'applications.
- Création des cas d'utilisation
  - En utilisant la notation UML, produire les cas d'utilisation du système logiciel. Le point de vue adopté est celui d'un généraliste. Autrement dit, ces cas d'utilisation doivent illustrer le principe de fonctionnement du système.

6

Planification et élaboration (4)

- En pratique:
  - La planification et l'élaboration des travaux sont des activités réalisées en collaboration avec le client, les experts du domaine d'application et les utilisateurs du système logiciel.
  - La spécification fournie par le client est souvent incomplète. Des rencontres doivent être prévues pour élucider les points obscurs contenus dans la spécification écrite par le client. Les experts du domaine d'application interviennent lorsque des questions techniques sont soulevées.

7

Développement itératif (1)

- Le point de départ:
  - Obtenir les exigences du logiciel à partir de la spécification fournie par le client.
  - Traduire ces exigences en termes de cas d'utilisation (diagrammes de cas d'utilisation et leur description).
  - Accorder une priorité aux cas d'utilisation.
  - Une stratégie pratique consiste à:
    - Accorder une grande priorité aux cas d'utilisation qui influencent grandement l'architecture du logiciel ou qui comportent des risques (technologiques ou financiers) non négligeables.

8

Développement itératif (2)

- Les tâches de cette phase d'activités:
  - Analyse orientée objet
    - Le but est de dégager le modèle conceptuel de la problématique à partir des cas d'utilisation.
    - Le point de vue est celui des utilisateurs.
    - Créer des diagrammes UML pour exprimer la structure statique et le comportement dynamique du système.
    - La question à poser est: Quelles sont les fonctionnalités du système ?
    - ON NE DOIT PAS CHERCHER « LE COMMENT FAIRE » MAIS BIEN DE TROUVER « LE QUOI FAIRE ».

9

Développement itératif (3)

  - Conception orientée objet
    - Le but est dégager une architecture du logiciel qui apporte une solution à la problématique.
    - Le point de vue est celui d'un ingénieur en logiciel.
    - Apporter des solutions informatiques aux problèmes soulevés lors de l'analyse orientée objet.
    - Reprendre les diagrammes UML (obtenus lors de l'analyse) et d'y ajouter des artifices informatiques nécessaires pour permettre la réalisation concrète.
    - La question à poser est: Comment réaliser les fonctionnalités exigées ?
    - ON DOIT CONCENTRER SUR « LE COMMENT FAIRE ».

10

Développement itératif (4)

- Dans la phase d'analyse:
  - Raffinement des cas d'utilisation
    - Il s'agit de revoir le (ou les) cas d'utilisation à traiter et apporter des modifications qui s'imposent.
  - Définir le modèle conceptuel
    - Il s'agit de faire ressortir les concepts véhiculés par les cas d'utilisation raffinés. Les concepts dans ce contexte sont employés comme synonyme de classes.
    - Donc, le modèle conceptuel est l'ensemble cohérent des objets exprimés par les cas d'utilisation en main.

11

Développement itératif (5)

  - Définir les diagrammes de séquence du système
    - Les objets sont en interaction. Nous utilisons les diagrammes de séquence pour illustrer leur comportement dynamique.
  - Définir le contrat des opérations
    - Les objets entrent en interaction et réalisent ses tâches à l'aide d'opérations. Un objet possède nécessairement des opérations. Le contrat est une énumération des opérations de chacun des objets. Les opérations sont identifiées par des phrases en langage naturel. Elles seront formalisées en langage de programmation dans la phase de conception.

12

Développement itératif (6)

- Définir le contrat des opérations (suite)
  - Une opération est identique à une fonction membre en C++ mais à ce stade-ci les opérations sont identifiées par des phrases en langage naturel.
  - *Note: Un objet sans opération n'a pas grande utilité.*
- Définir les diagrammes d'états
  - Les objets sont des entités dynamiques et ils possèdent des états. Ces diagrammes explicitent les états des objets. Il s'agit d'une façon succincte de présenter les opérations des objets et l'effet de ces opérations sur les états des objets.

13

Développement itératif (7)

- Raffiner le dictionnaire des termes
  - Le processus de développement apporte inévitablement de nouveaux concepts et nouvelles entrées dans le dictionnaire des termes.
  - Il est primordial d'effectuer une mise à jour du dictionnaire dans chacune des itérations du processus de développement.

14

Développement itératif (8)

- Dans la phase de conception
  - Raffiner les cas d'utilisation
    - Dans la phase de conception, le but est de rechercher une solution à la problématique. Les cas d'utilisation doivent être raffinés afin de faire ressortir les concepts qui appartiennent au domaine de la solution.
  - Définir la présentation du logiciel
    - Le but de cette activité consiste à décrire d'une manière logique l'interface de présentation du logiciel. Dans la plupart des cas, cela signifie la représentation de l'interface graphique.

15

Développement itératif (9)

  - Raffiner l'architecture du système
    - L'architecture du système logiciel doit évoluer jusqu'à la " maturité " ou jusqu'à une certaine stabilité. Cette évolution est obtenue en la raffinant à chaque itération du processus de développement.
  - Définir les diagrammes de classes
    - C'est à partir des concepts (classes) dégagés des cas d'utilisation que l'on peut identifier les classes. Le fruit de ce travail fort important est exprimé sous forme de diagrammes de classes.

16

Développement itératif (10)

- Définir les bases de données
  - La plupart des applications d'envergure utilisent des bases de données. C'est dans la phase de conception que l'on conçoit les tables et les liens de ces bases de données.
- Dans la phase de construction
  - La construction est le codage proprement dit du système logiciel.
  - L'écriture du code source n'est pas une activité isolée. Elle s'appuie plutôt sur le travail réalisé dans les phases d'analyse et de conception.

17

Développement itératif (11)

- Dans la phase de construction (suite)
  - La phase de la construction comprend également le déverminage afin d'éliminer, dans la mesure du possible, les erreurs syntaxiques et logiques du code source.
  - De plus en plus, le déverminage est réalisé par des spécialistes qui ont pour tâches l'examen du code source.

18

Développement itératif (12)

- Le code source doit également être documenté. Au minimum, le code source est commenté par le programmeur.
- Dans la plupart des cas, un cahier de notes est utilisé pour expliquer plus clairement l'implantation des algorithmes et la définition des structures de données.
- Chaque fichier source doit être contrôlé. Ce contrôle est représenté par un numéro de version. Le numéro de version est attribuable manuellement ou automatiquement par un système de gestion. (ex: CVS à http://www.cvshome.org)

19

Développement itératif (13)

- La phase de la validation
  - Dans notre contexte, renferme les activités reliées aux tests et aux mesures de la performance du système logiciel.
  - Les tests de logiciel concernent les tests d'unité, les tests d'intégration et du système.
  - Test d'unité
    - Il s'agit d'éliminer, dans la mesure du possible, les erreurs de codage et de fonctionnement du système logiciel.

20

Développement itératif (14)

- La phase de la validation (suite)
  - Test d'intégration et de système
    - Placer le système logiciel dans son environnement d'utilisation et réaliser des tests afin de déceler les erreurs de fonctionnement du système logiciel.
    - Une attention particulière est accordée aux interfaces du système logiciel avec les autres sous-systèmes de son environnement d'utilisation.

21

Développement itératif (15)

- La phase de validation (suite)
  - Test de performance
    - Il s'agit de valider les limites de performance du système logiciel selon les spécifications du client.
    - L'accent sur mis sur la performance et la fiabilité du système logiciel. Les résultats obtenus serviront à raffiner les algorithmes et les structures de données utilisées dans le système logiciel.

22

Le déploiement (1)

- Le déploiement est l'application du système logiciel dans son environnement d'utilisation.
- Il constitue la mise en place du logiciel chez le client ou l'adoption du système comme outil opérationnel.
- Voici les activités normalement associées à cette étape :
  - Documentation technique
  - Rassembler tous les diagrammes UML et documents techniques reliés au processus de développement du système logiciel.

23

Le déploiement (2)

– Manuel d'utilisation
  - Présenter l'utilisation du système logiciel en langage naturel.
  - Parfois, il faut présenter le système selon le point de vue du gestionnaire (du système) et selon le point de vue d'utilisateurs qui entrent en interaction avec le système.
  - L'écriture du manuel d'utilisateur est sans doute la partie la plus négligée de cette étape.

24

Le déploiement (3)

- Test d'intégration et de système
  - Ces tests sont réalisés dans le milieu de fonctionnement du système. Il est donc normal que l'étape de déploiement soit aussi impliquée dans le test d'intégration et de système.
- Apprentissage et support
  - Il s'agit de prévoir l'enseignement de l'utilisation du système par des séances d'apprentissage.
  - Le déploiement d'un système logiciel avec séances d'apprentissage peut augmenter l'acceptation du logiciel par ses utilisateurs.

25

Le déploiement (4)

- Apprentissage et support (suite)
  - C'est aussi lors de la planification du déploiement que l'on établit la logistique de support.
  - Le personnel au sein de l'entreprise entraîné ou des techniciens de la maison de production du logiciel peut jouer le rôle de personnel de support du système logiciel.

26

Utilisation des cadres de travail (1)

- Quelques cadres de travail
  - Windows
    - MFC de Microsoft
    - Visual Components (Borland)
    - etc.
  - MacOS
    - PowerPlant de CodeWarrior
    - etc.
  - UNIX
    - Visual Workshop de SunSoft
    - Ktk pour LINUX/KDE
    - etc.

Bref, de nos jours l'utilisation des cadres est inévitable.

27

Utilisation des cadres de travail (2)

- L'utilisation de ces cadres de travail (et des classes préfabriquées) produit parfois des problèmes de « réutilisation » et de « portabilité ».
- L'application réalisée est fortement couplée aux cadres de travail.
  - Par exemple, la classe **`CWnd`** de MFC n'existe pas dans le Visual Components.
- On ne peut insérer facilement nos classes dans un autre cadre de travail.
- On ne peut implanter facilement nos classes sur une autre plate-forme.

28

Utilisation des cadres de travail (3)

- Les générateurs de code de ces cadres de travail tentent à produire du code taillé sur mesure.
  - Un facteur qui contribue à un couplage serré entre l'application et le cadre de travail utilisé.
- Il y a confusion entre l'interface graphique et l'application.
- Pour les programmeurs novices: l'interface graphique EST l'application.
  - Le codage est réalisé directement parmi les éléments de l'interface graphique. (Oupps ...)

29

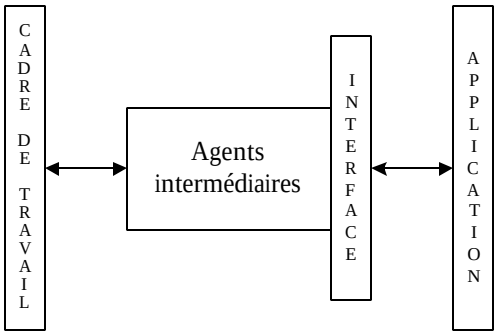
Utilisation des cadres de travail (4)

- Le couplage serré entre l'application et l'interface graphique donne des difficultés à l'entretien et l'évolution du logiciel.
  - Le code et les algorithmes de l'application sont dispersés parmi les différents contrôles de l'interface graphique.
  - Les changements de l'interface graphique peuvent forcer la modification des algorithmes de l'application !

30

Utilisation des cadres de travail (5)

- Dans la pratique:
  - Nous pouvons utiliser le concept d'agent intermédiaire pour réduire le couplage entre le cadre de travail et l'application.



Algorithmes, structures de données, classes et objets de l'application, etc.

31

Persistance des objets (1)

- Le besoin de stockage est normalement dégagé lors de l'analyse des exigences.
- Dans le cas de stockage simple de données, on peut utiliser les méthodes traditionnelles.
  - Fichiers de données
  - Base de données
- Dans le cas de stockage d'objets, il est nécessaire de considéré les alternatifs suivants:
  - Utiliser les services offerts par le cadre de travail
    - Ex: Serialisation de MFC, Streamable Objects de Borland

32

Persistance des objets (2)

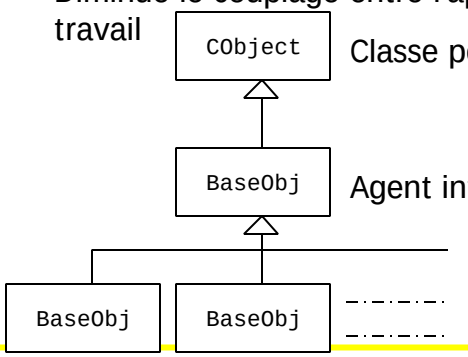
- Créer nous-mêmes les services offrant la persistance des objets.
- Utiliser des agents intermédiaires.

- Avantages et désavantages
  - Utiliser les services du cadre de travail
    - Utilisation rapide, code robuste et professionnel
    - Fortement couplé au cadre de travail
  - Créer nous-mêmes les services offrant la persistance des objets.
    - Développement complexe → coût élevé
    - Réutisable et portable

33

Persistance des objets (3)

- Avantages et désavantages
  - Utiliser des agents intermédiaires.
    - Solution mitoyenne
    - Diminue le couplage entre l'application et le cadre de travail



Classe persistance de MFC

Agent intermédiaire

Classe persistance de l'application

34

Fin du chapitre 7

- Lire les notes de cours du chapitre 7.
- Pratiquer la création des diagrammes UML en utilisant les recommandations contenues dans ce chapitre.

N'oubliez pas: C++ est bien plus que du C.

35